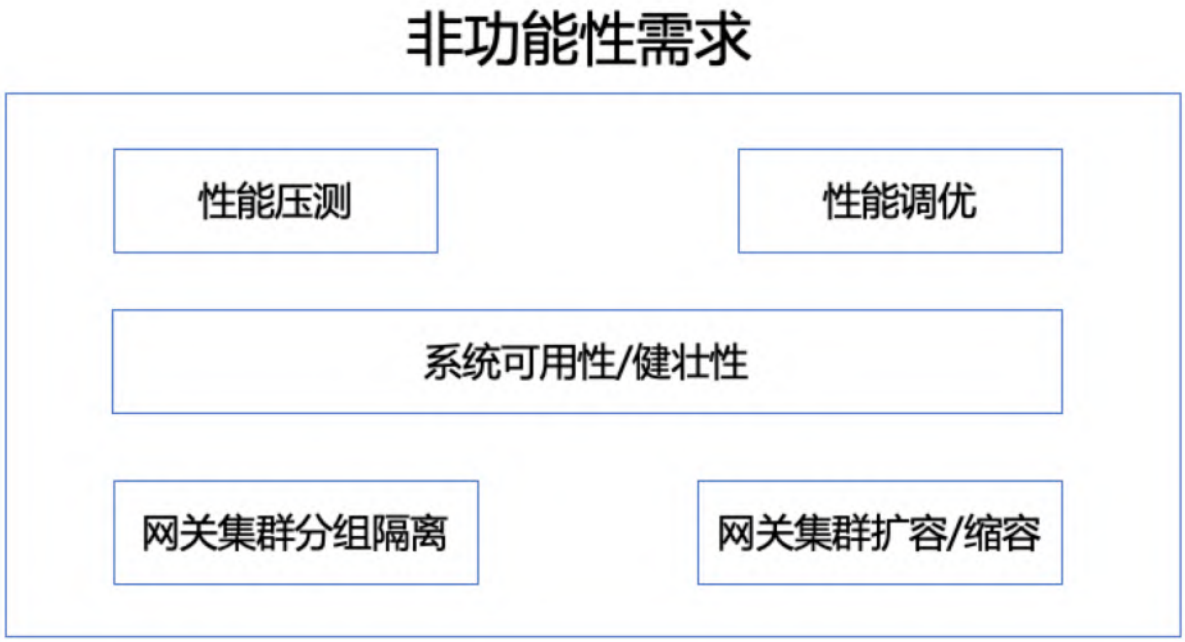
API网关架构设计V1.0

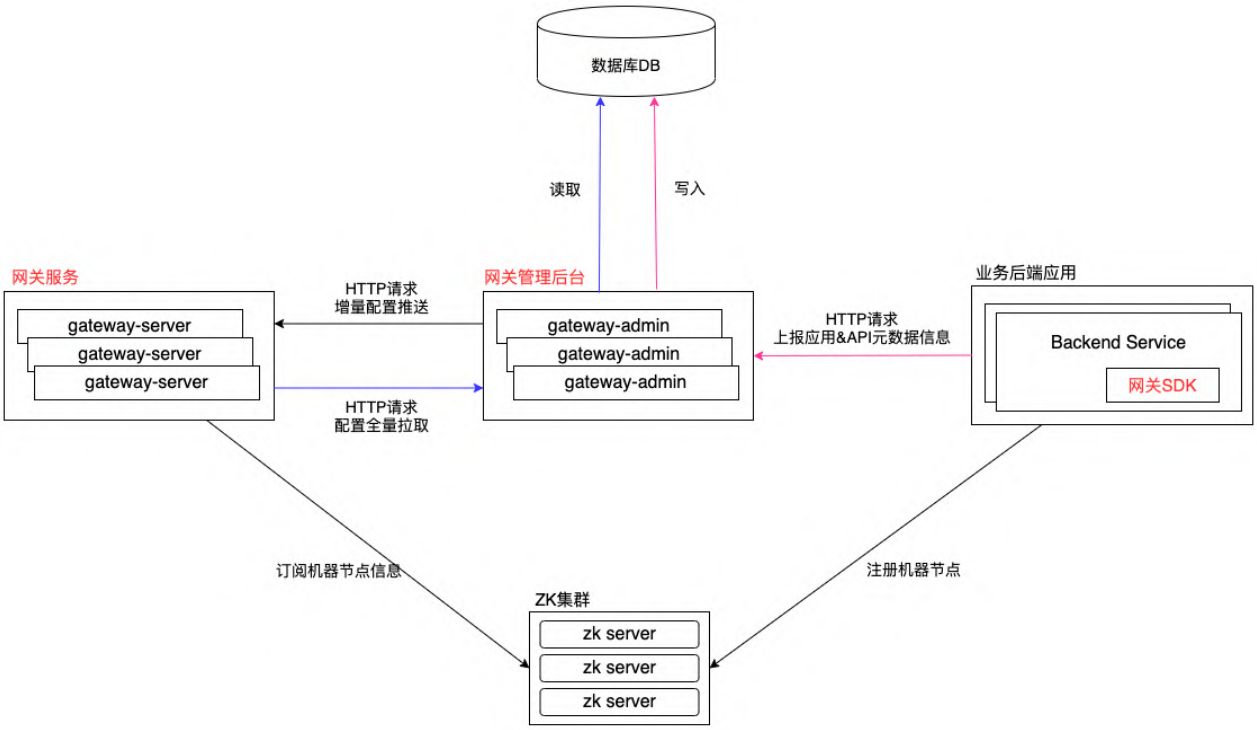
功能性需求

非功能性需求

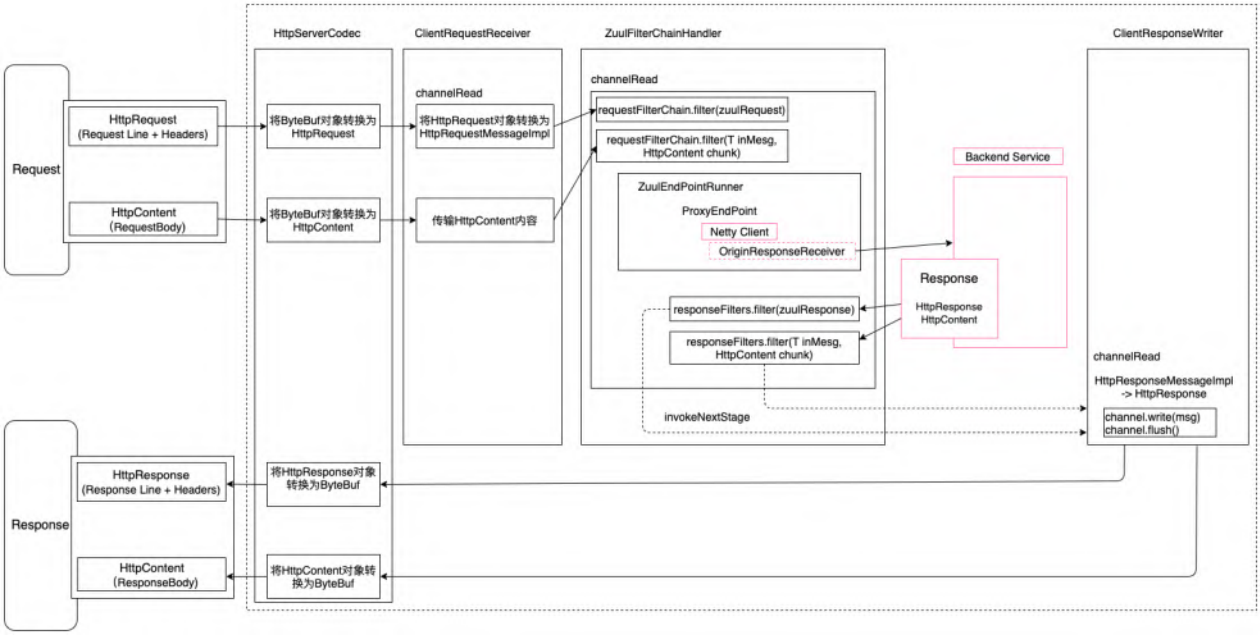


模块划分

- 网关服务
- 网关管理后台
- 网关SDK



网关服务



如上图所示，Netty自带的HttpServerCodec会将网络二进制流转换为Netty的HttpRequest对象，再通过ClientRequestReceiver编解码器将HttpRequest转换为Zuul的请求对象HttpRequestMessageImpl；

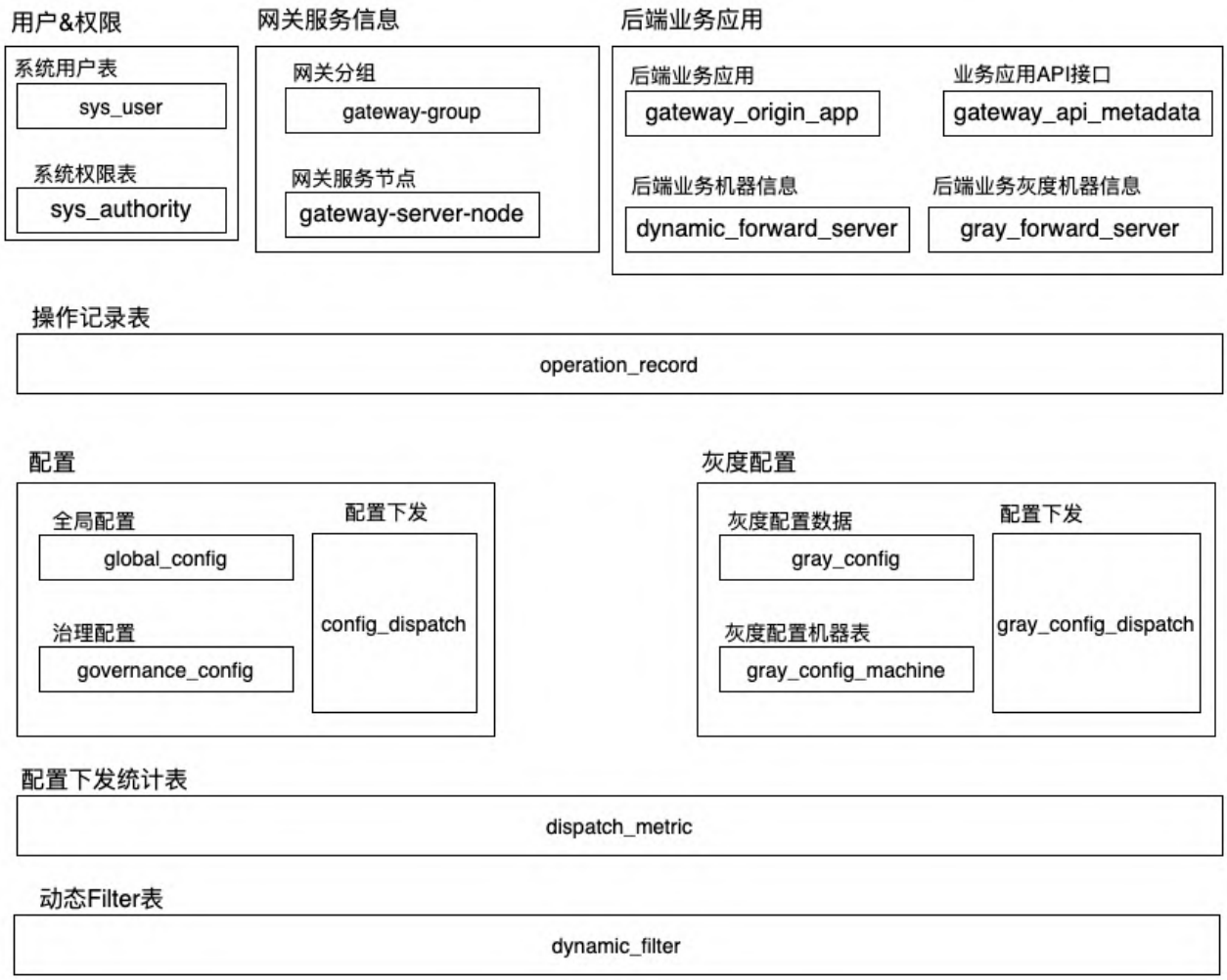
请求体RequestBody在Netty自带的HttpServerCodec中被映射为HttpContent对象，ClientRequestReceiver编解码器依次接收HttpContent对象。

完成了上述数据的转换之后，就流转到了最重要的编解码ZuulFilterChainHandler，里面会执行Filter链，也会发起网络请求到真正的后端服务，这一切都是在ZuulFilterChainHandler中完成的

得到了后端服务的`响应结果`之后，也经过了Outbound Filter的过滤，接下来就是通过ClientResponseWriter把Zuul自定义的响应对象HttpResponseMessageImpl转换为Netty的HttpResponse对象，然后通过HttpServerCodec转换为ByteBuf对象，发送网络二进制流，完成响应结果的输出。

网关管理后台

管理后台模块



数据库表设计

数据库设计如下：

- 治理配置表`governance_config`
  - 用途：为应用、API接口设置治理信息，比如限流配置、熔断配置、超时时间
- 全局配置项`global_config`
  - 用途：黑名单信息、全局开关
- Filter下发表`dynamic_filter`
- 配置下发历史表`config_dispatch`
- 后端应用表`gateway_origin_app`

- API元数据表`gateway_api_metadata`
- 后端服务机器表`dynamic_forward_server`
- 灰度后端服务机器表`gray_forward_server`
- 网关分组表`gateway_group`
- 网关集群节点表`gateway_server_node`
- 灰度配置表`gray_config`
- 机器灰度配置表`gray_config_machine`
- 配置下发历史表`gray_config_dispatch`
- 配置下发统计表`dispatch_metric`
- 系统用户表`sys_user`
- 系统权限表`sys_authority`
- 操作记录表`operation_record`

具体的数据库ER图如下：

数据库SQL -> [gateway-admin.sql]

网关SDK

工程名为`gateway-client`

配置项：

- `gateway.client.api.scan.auto`：是否开启API自动上报
- `gateway.client.registry.address`：注册中心地址
- `gateway.client.app.name`：应用名

提供注解：

- `@GatewayApi`：标记到Method上面，用于主动上报要注册到网关的API接口
- `@GatewayApiIgnore`：在开启`API自动上报`的同时，用于`忽略`某些API，这些API不需要注册到网关

接入类型：

- dubbo接入
  - `gateway-client-alibaba-dubbo`：自定义`BeanPostProcessor`，用于提取到`ServiceBean`，主动上报到网关后台
  - `gateway-client-apache-dubbo`：自定义`ApplicationListener`，用于监听`ServiceBeanExportedEvent`事件，提取`event`信息，上报到网关后台
- `gateway-client-spring-mvc`、`gateway-client-spring-boot`接入
  - 自定义`BeanPostProcessor`，用于提取到`Controller`、`RestController`的`RequestMapping`注解，放入线程池异步上报API信息
  - 自定义`ApplicationListener`，监听`WebServerInitializedEvent`事件，用于注册`应用信息到注册中心`

备注：如果一个项目同时使用了alibaba-dubbo和spring-boot，则同时依赖`gateway-client-alibaba-dubbo`、`gateway-client-spring-boot`即可

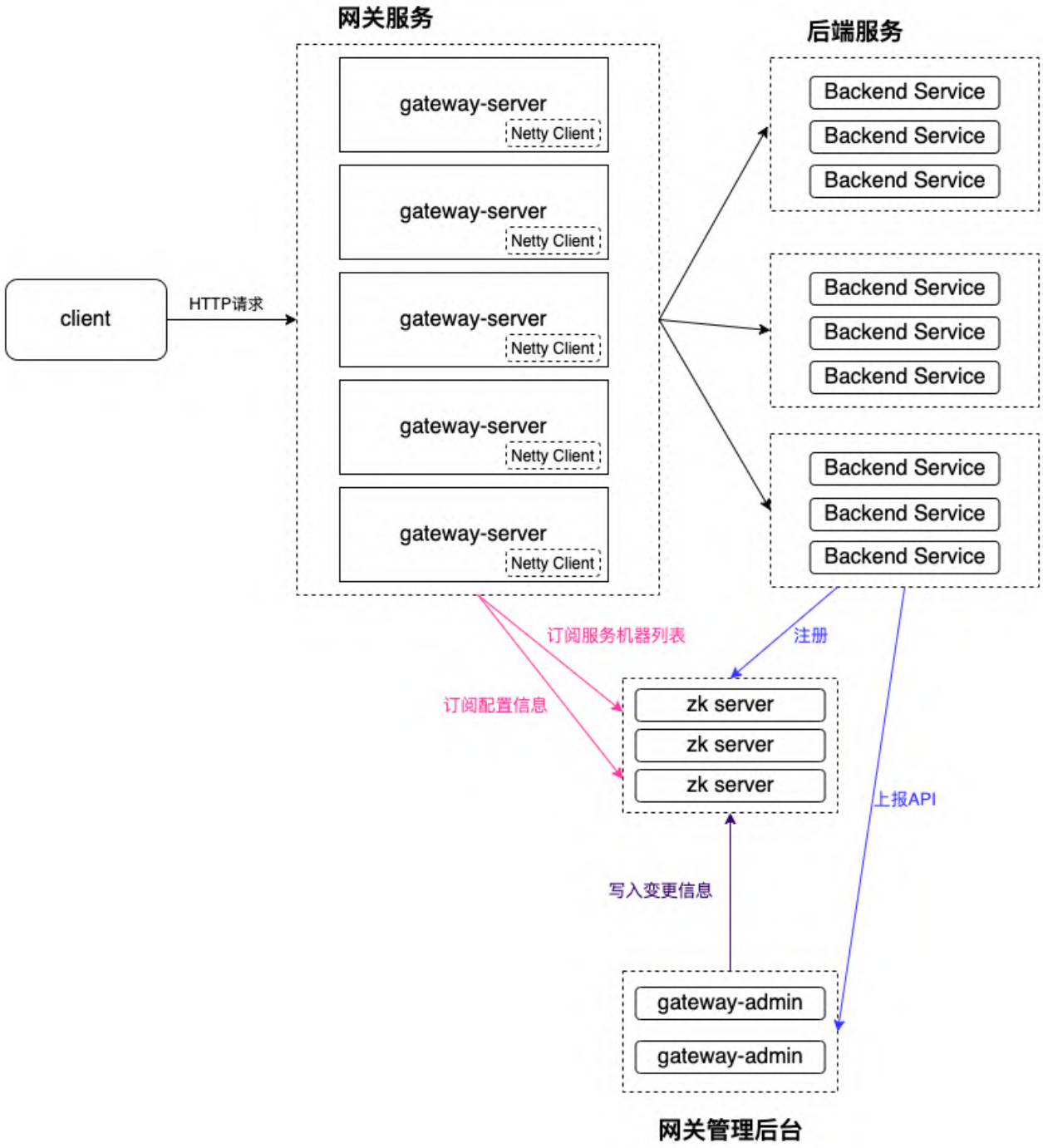
通信机制

由于需要动态的下发配置，比如全局开关、应用的治理配置、接口级别的治理配置，就需要网关管理后台可以与网关服务进行通信，比如推拉模式。

两种设计方案

- 1.基于注册中心的订阅通知机制
- 2.基于HTTP的推模式

基于注册中心的订阅通知机制



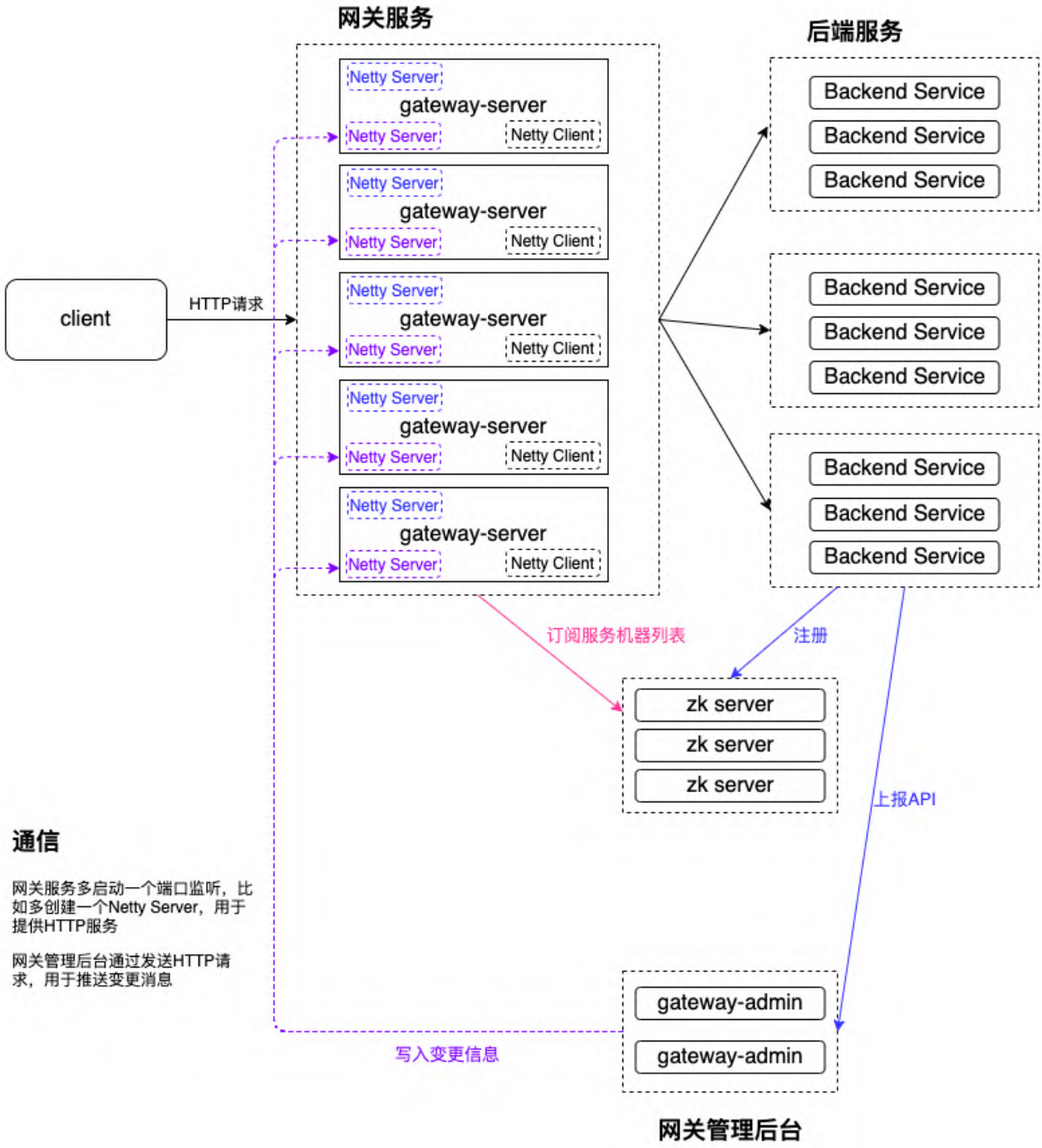
这是个可行的方案，但是存在以下几个缺点：

- 严重依赖zk集群的稳定性
- 信息不私密(目前公司级的zk集群没有权限管控、可随意查看、可随意删除)
- 无法灰度下发配置，比如只对其中的一台`网关服务`配置生效

上面的问题部分是可解决的，但是如果为了解决这些问题，而把问题复杂化的话，就得不偿失了。

为此，我们摒弃了上述的方案，改为`基于HTTP的推模式`

基于HTTP的推模式



全量配置拉取

全量拉取的逻辑

- 1.创建ZkClient，用于全量拉取存储于zookeeper的`后端服务机器信息`
- 2.发起HTTP请求到`网关管理后台`，全量拉取归属于自己的配置信息

在完成了上面的两件事情之后，还得发起HTTP请求登记自身信息到`网关管理后台`

这样`网关管理后台`就可以提取到全部的`网关服务`列表了

灰度配置下发

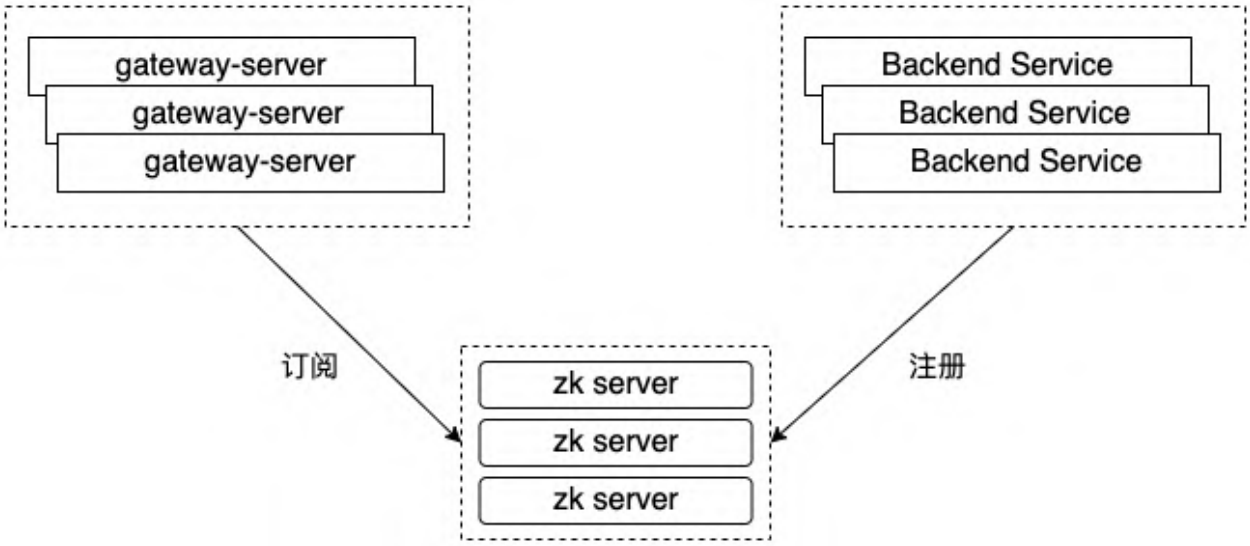
完成上面说的`全量配置拉取`、`网关服务登记`之后，`网关管理后台`可以获取到全部的`网关服务`列表

当编辑了某个接口的治理配置信息之后，比如限流信息、超时时间，就可以选中`网关服务`的一台，先进行推送，推送完成之后，观察一段时间，再全量推送。而推送的逻辑也很简单，只需要发起HTTP请求到`网关服务`，如果发现推送失败，则重新推送。

`网关管理后台`与`网关服务`之间的HTTP通信，会采用签名的方式验证请求的合法性，校验失败的话，则无法进行通信。

1.服务注册发现

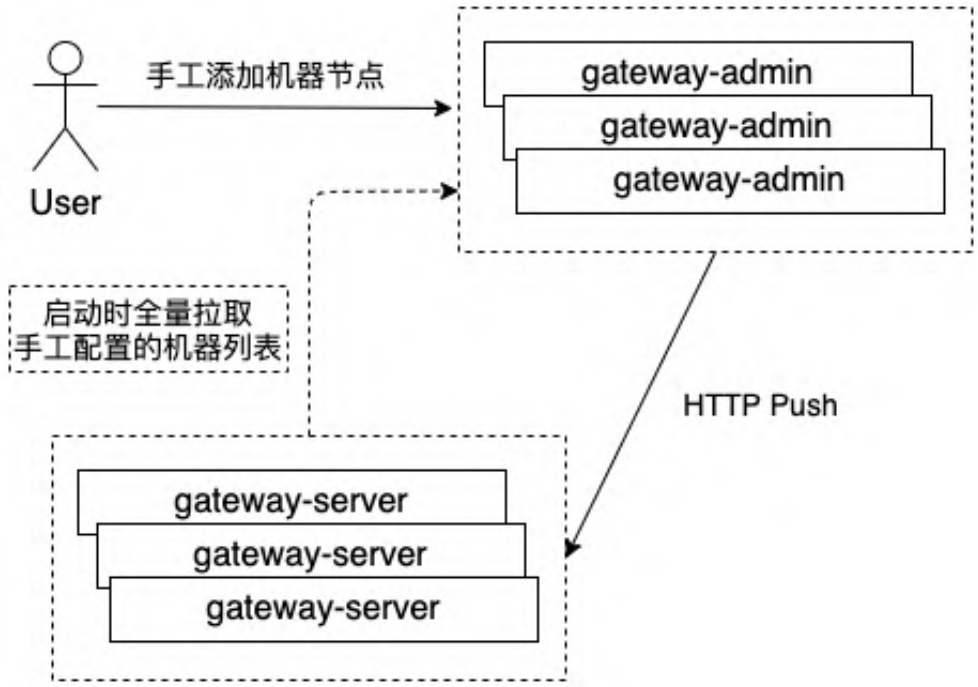
基于zookeeper的订阅发现



后端服务`Backend Service`会在启动的时候往zookeeper注册当前的机器信息，是临时节点，如果zk session会话过期，则会自动删除

网关服务`gateway-server`会订阅后端服务机器信息，如果有变更，则会立马得到通知，并刷新当前的机器节点列表。

手工添加推送



如果有用户不希望依赖`gateway-client`，也不想在zookeeper上注册机器节点信息，但是又想使用网关的功能，这个时候，网关就需要支持手工添加机器节点

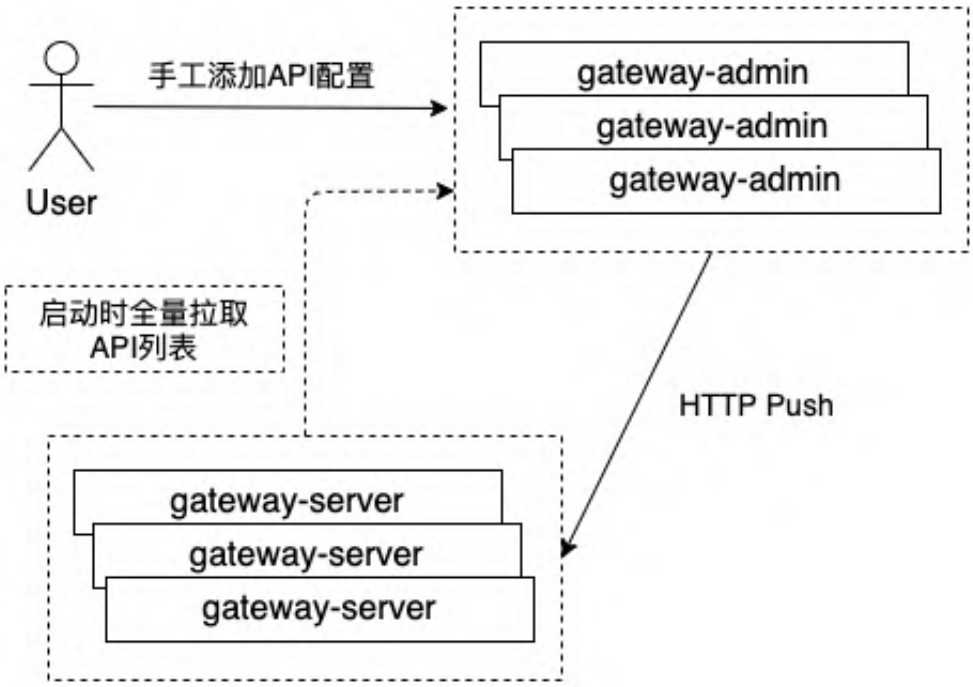
用户通过网关管理后台`gateway-admin`添加机器节点，然后网关管理后台就会异步推送增量的机器信息给网关服务`gateway-server`

那么上述的方案也只是解决了增量推送的问题，那么肯定需要网关服务`gateway-server`在启动之初就全量拉取一次全部的手工添加节点列表。

2.API管理

手动管理

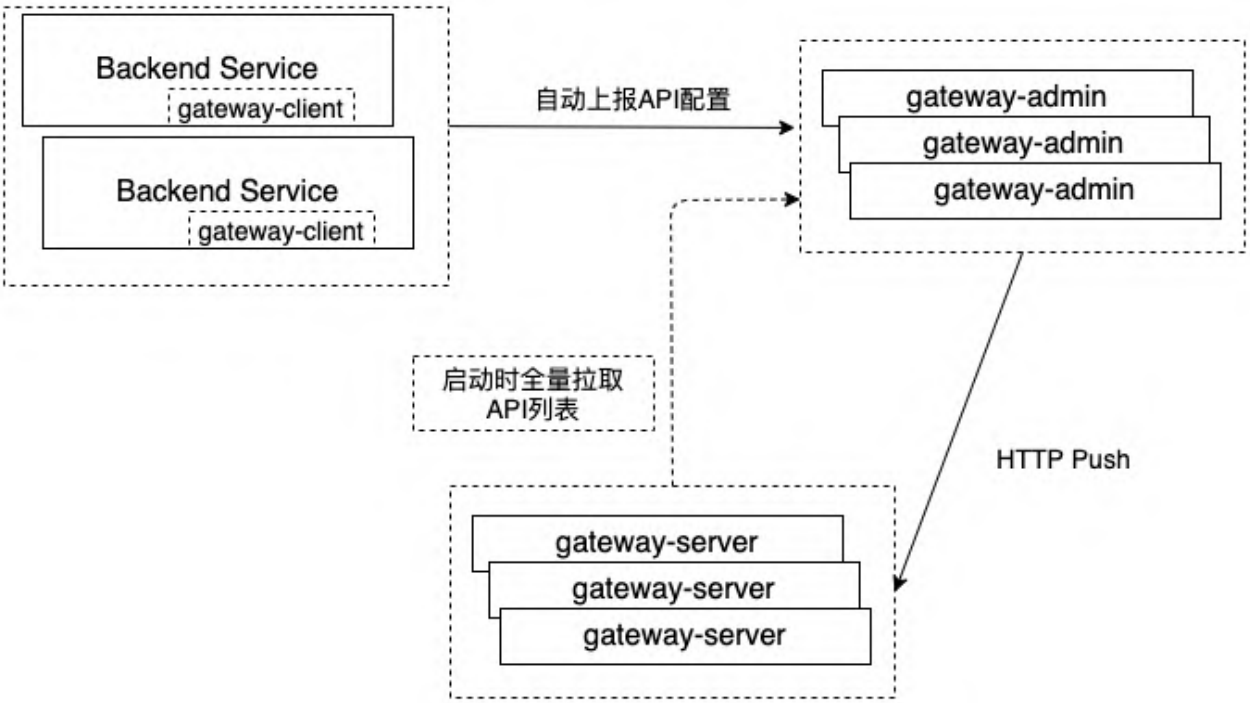
部分业务不愿意依赖网关SDK`gateway-client`，但是也希望使用网关的功能，为此需要预留`手动管理`的功能



自动上传

后端服务依赖网关SDK`gateway-client`，通过扫描注解会放入线程池，发起网络请求，自动上报到网关管理后台

网关管理后台再将这些`API元数据`异步推送给`网关服务`，如果推送失败，则写入`推送记录表`，后续进行重试。

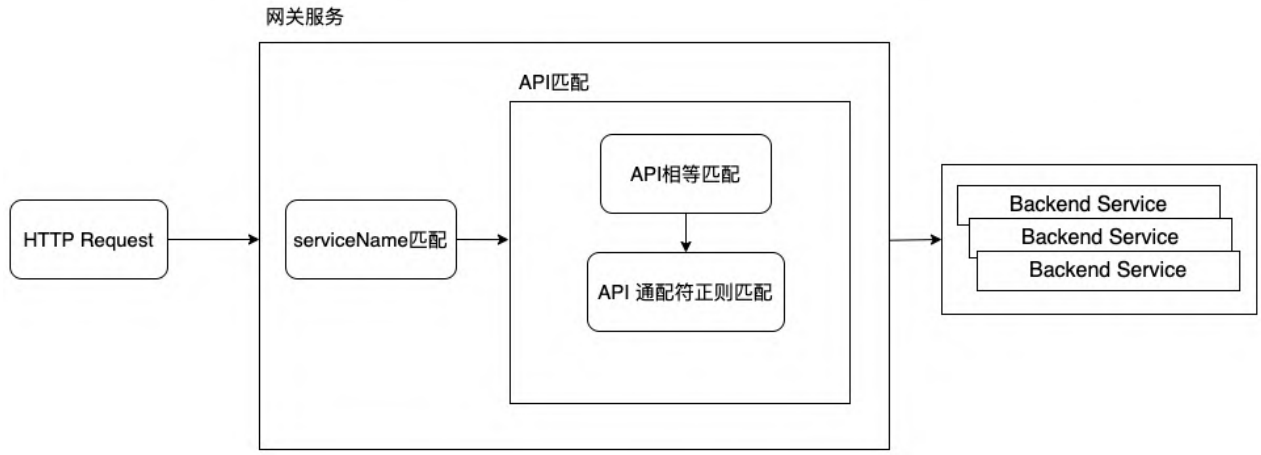


API映射

网关在进行请求转发的时候，需要明确知道请求哪一个服务的哪一个API，这个过程就是API匹配。

因为不同的后端服务可能会拥有相同路径的API，所以网关要求用户必须传递`serviceName`，服务名可以放置于请求Header或者请求参数中

携带了serviceName之后，就可以在后端服务的API中去匹配了，有一些是相等匹配，有些是正则匹配，因为RESTFul协议，需要支持`/**`通配符匹配。



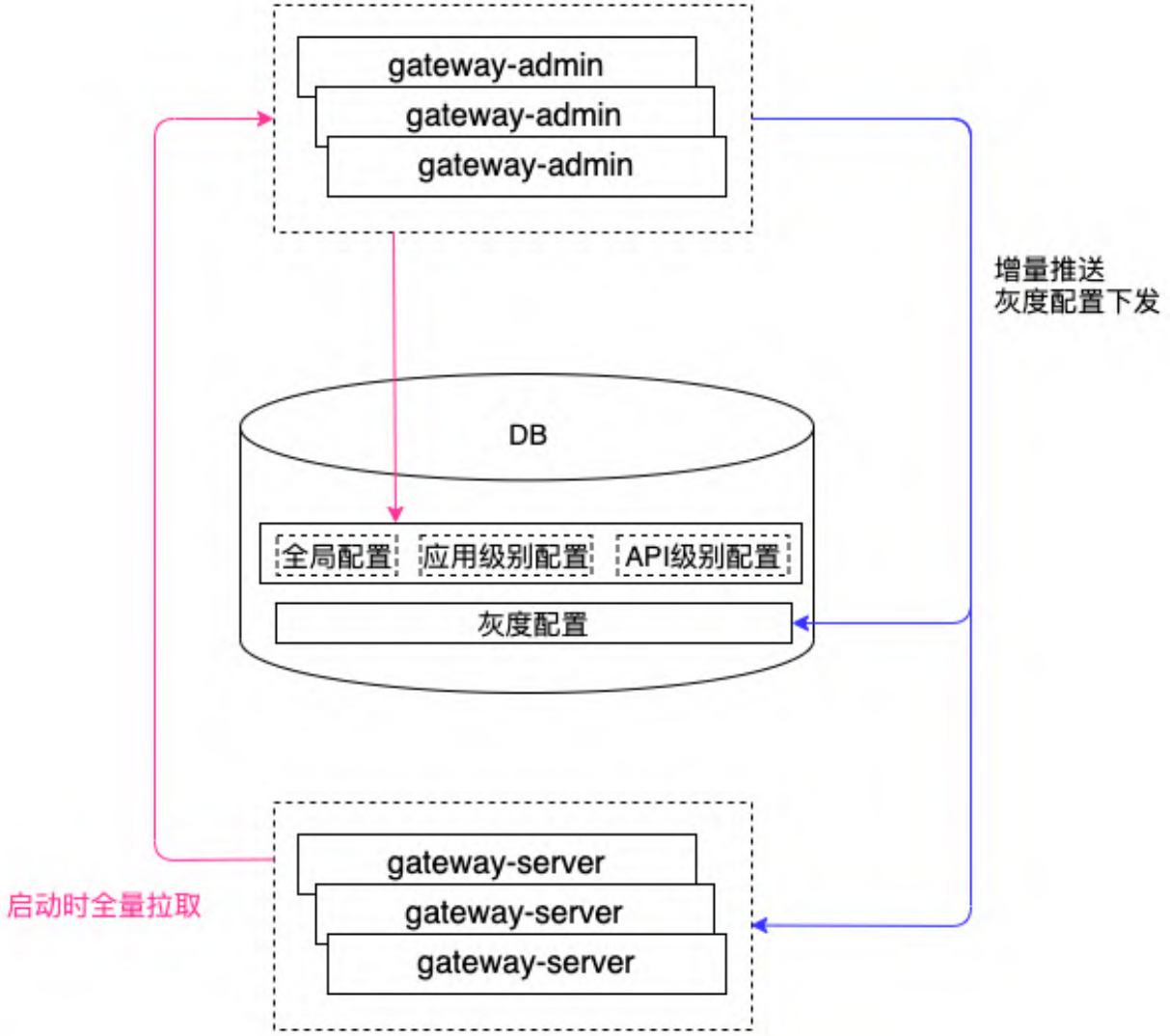
3.动态配置

Zuul依赖的动态配置为archaius，而archaius的动态更新依赖`FixedDelayPollingScheduler`

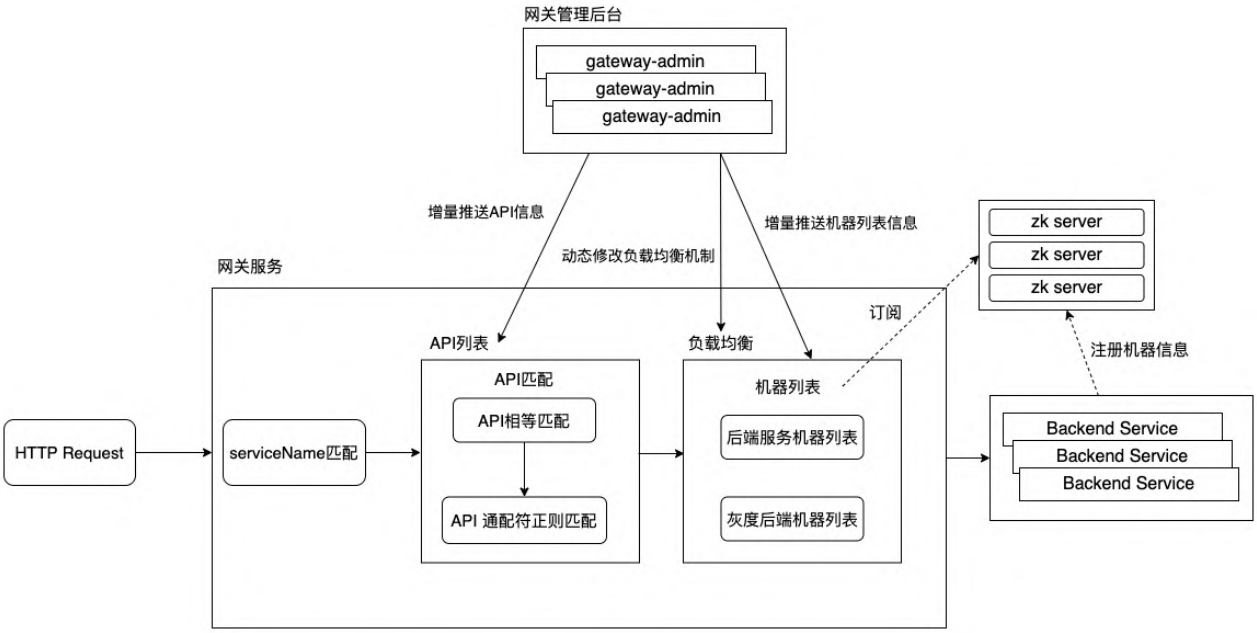
可以通过扩展`com.netflix.config.DynamicConfiguration` + `FixedDelayPollingScheduler`从而实现整合vivo中间件配置中心的功能。

然而经过衡量之后，放弃了这个想法，因为`vivo中间件配置中心`存在两个线程，而网关是线程敏感型服务，尽量减少线程的使用。并且已经有HTTP Push的功能，只需要在`HTTP Push`的时候，往`archaius`内部写入增量的key/value配置项即可。

同时在网关服务启动的时候，会全量拉取一次配置，将这些全量的key/value配置项写入`archaius`


gateway-admin
gateway-admin
gateway-admin
增量推送
灰度配置下发
DB
全局配置
应用级别配置
API级别配置
灰度配置
gateway-server
gateway-server
gateway-server
启动时全量拉取

4.动态路由

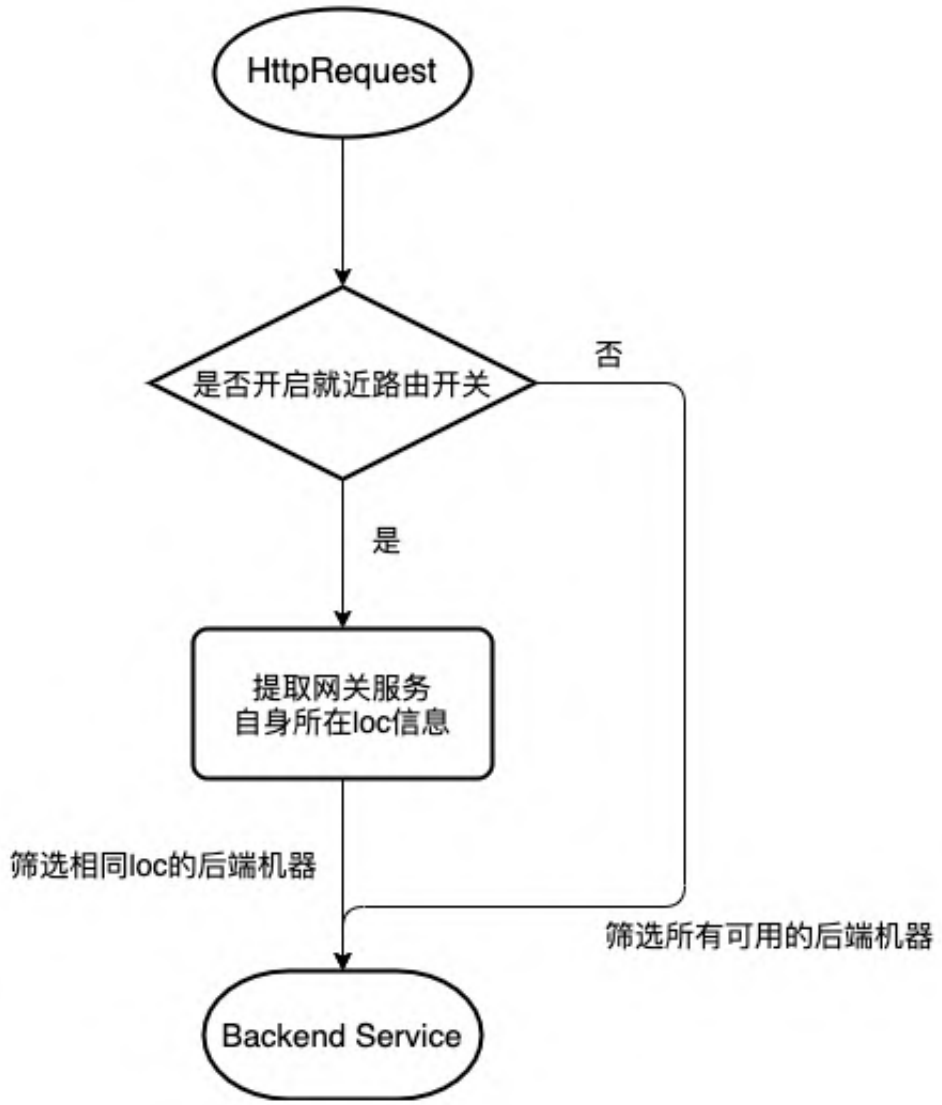

网关管理后台
gateway-admin
gateway-admin
gateway-admin
增量推送API信息
动态修改负载均衡机制
增量推送机器列表信息
zk server
zk server
zk server
网关服务
订阅
注册机器信息
API列表
负载均衡
API匹配
机器列表
HTTP Request
serviceName匹配
API相等匹配
API 通配符正则匹配
后端服务机器列表
灰度后端机器列表
Backend Service
Backend Service
Backend Service

4.1 就近路由

就近路由也就是同机房路由

当请求到`网关服务`，会提取网关服务自身的机房`loc`属性值，读取全局、应用级别的开关，如果就近路由开关打开，则筛选服务列表的时候，会过滤相同`loc`的后端机器，负载均衡的时候，在相同`loc`的机器列表中挑选一台进行请求。

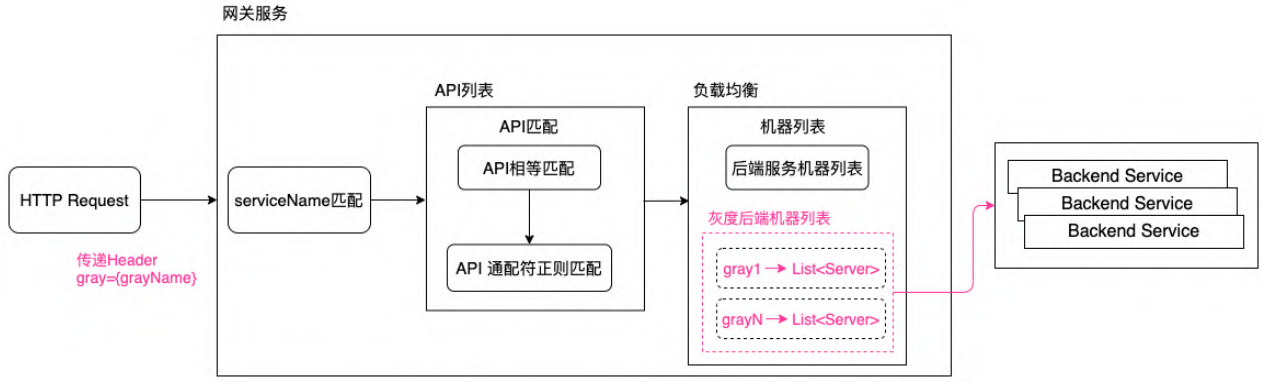
如果没有相同`loc`的后端机器，则降级从`其他loc`的后端机器中进行挑选。



4.2 灰度路由

灰度路由需要用户传递Header属性值，比如`gray=canary_gray`

网关管理后台配置灰度路由的时候，会设置`grayName -> List<Server>`，当网关管理后台增量推送到网关服务之后，网关服务就可以通过grayName来提取配置下的后端机器列表，然后再进行负载均衡挑选机器。



5.负载均衡

Zuul使用Ribbon组件来实现负载均衡，如果想要定制的话，需要继承`DynamicServerListLoadBalancer`

举例：

```
#假设后端服务名originName=originA
originA.ribbon.listOfServers=localhost:8801,localhost:8802
originA.ribbon.NFLoadBalancerClassName=com.netflix.loadbalancer.ZoneAwareLoadBalancer
```

网关服务通过修改`{originName}.ribbon.listOfServers`即可修改负载均衡的机器列表

网关服务通过修改`{originName}.ribbon.NFLoadBalancerClassName`即可动态切换负载均衡算法

默认的负载均衡算法为ZoneAwareLoadBalancer，而ZoneAwareLoadBalancer依赖的是AvailabilityFilteringRule

AvailabilityFilteringRule：会先过滤掉由于多次访问故障而处于断路器状态的服务，还有并发的连接数量超过阈值的服务，然后对剩余的服务列表按照轮询策略进行访问

心跳检测

核心思路：当网络请求正常返回的时候，心跳检测是不需要，此时后端服务节点肯定是正常的，只需要定期检测未被请求的后端节点，超过一定的错误阈值，则标记为不可用，从负载均衡列表中剔除。

6.协议转发

HTTP -> HTTP

Zuul采用的是ProxyEndpoint用于支持`HTTP -> HTTP`协议转发

通过Netty Client的方式发起网络请求到真实的后端服务

HTTP -> Dubbo

采用Dubbo的泛化调用实现`HTTP -> Dubbo`协议转发

样例代码：

```
//com.dubbo.study.hello.api.HelloService
public interface HelloService {

    String sayHello(String name);

    String printMulti(String name, String phone);

    ComplexElementDto sayComplex(ComplexElementDto param);

    ComplexElementDto sayComplex2(ComplexElementDto param, SysUserDto user);

}
```

简单数据类型_单参数/多参数 泛化调用

```
// 当前应用配置
ApplicationConfig application = new ApplicationConfig();
application.setName("generic-reference-app");

// 连接注册中心配置
RegistryConfig registry = new RegistryConfig();
registry.setProtocol("zookeeper");
registry.setAddress("zookeeper://10.101.92.28:2183");

// 引用远程服务
ReferenceConfig<GenericService> reference = new
ReferenceConfig<GenericService>();
reference.setInterface("com.dubbo.study.hello.api.HelloService");
reference.setApplication(application);
reference.setRegistry(registry);
reference.setRetries(0);
reference.setGeneric(true);

GenericService genericService = reference.get();

// 单参数泛化调用
Object result = genericService.$invoke("sayHello", new String[]
{"java.lang.String"}, new Object[]{"World"});

// 多参数泛化调用
Object resultTwo = genericService.$invoke("printMulti"
```

```
        , new String[]{"java.lang.String", "java.lang.String"}
        , new Object[]{"World", "1234567890"});
```

复杂数据类型_多参数 泛化调用

```
// 多参数复杂类型_泛化调用

Map<String, Object> params = new HashMap<>();
Map<String, Object> basicParam = new HashMap<>();
basicParam.put("basicInt", 1);
basicParam.put("basicString", "test");
params.put("basic", basicParam);

Map<String, Object> user = new HashMap<>();
user.put("name", "lin");
user.put("phone", "123456789");

Object complexResult = genericService.$invoke("sayComplex2"
  , new String[]{"com.dubbo.study.hello.dto.ComplexElementDto",
"com.dubbo.study.hello.dto.SysUserDto"}
    , new Object[]{ params, user});
```

借鉴上述的样例代码，可以实现一个DubboForwardEndpoint，内部通过泛化调用来实现协议转换，GenericService需要缓存在内存中，避免频繁的创建与销毁。

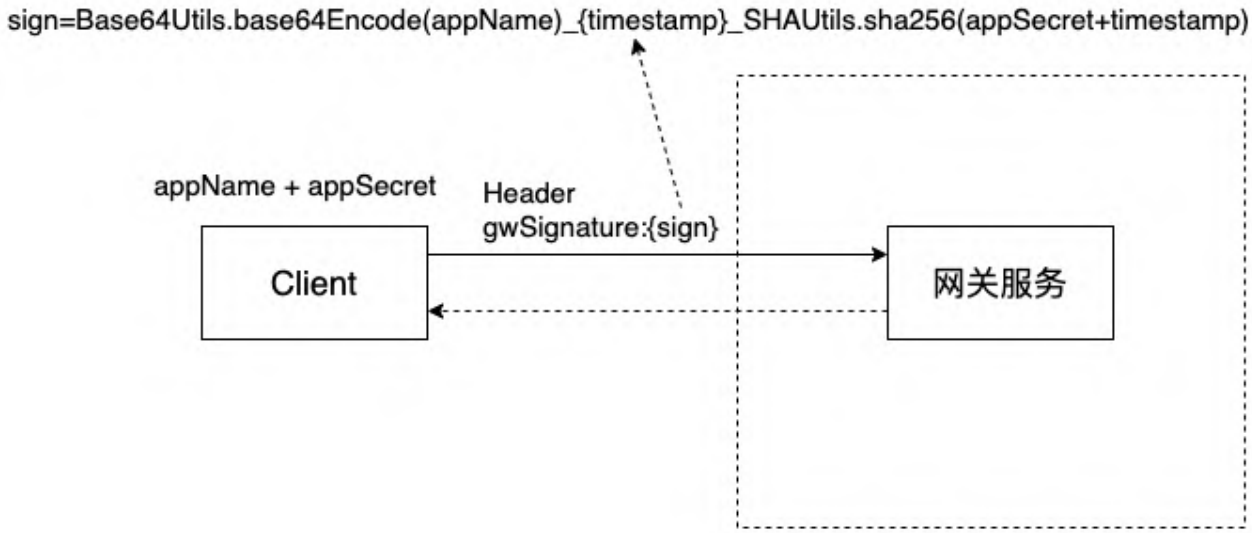
7.安全机制

IP黑名单

基于Netty Handler扩展，增加IpBlacklistHandler，通过全局配置下发黑名单信息，实现动态拦截

appName/appSecket验签

基于Netty Handler扩展，增加AppSignatureHandler，拦截Header中的签名是否合法，如果合法，则请求继续流转



验签细节：

- 请求方在`网关管理后台`获取应用的appName&appSecret
- Client请求`网关服务`，需要传递Header，Header的key为gwSignature，value为`Base64Utils.base64Encode(appKey)_{timestamp}_SHAUtils.sha256(appSecret+timestamp)`
- 安全通信的关键在于SHA256算法的不可逆+时间戳有效期，参数传递的timestamp不能与服务器的时间戳相差超过1小时
- 网关服务添加AppSignatureHandler，拦截请求，提取appName，获取对应的appSecret，计算SHA256哈希值，校验是否相等

矛盾加解密

金融网关已支持矛盾加解密，后续会整合进来

8.监控/告警

访问日志Access Log

Zuul已实现Access Log日志记录功能，与Nginx、Tomcat的`AccessLog`类似

关联的类如下：

- com.netflix.netty.common.accesslog.AccessLogChannelHandler
- com.netflix.netty.common.accesslog.AccessLogPublisher

原理如下：

将`AccessLogChannelHandler`织入pipeline中

- 识别到`channelRead`事件之后，创建一个`RequestState`对象，记录当前的请求信息、请求时间，放入到Channel的`Attribute`属性中
- 识别到`HttpLifecycleChannelHandler.CompleteEvent`之后，从Channel的`Attribute`属性拿出`RequestState`对象记录当前的时间戳；通过`AccessLogPublisher#log`打印日志

这里记录的耗时为总耗时，并不能计算出网关导致的耗时时延。

需要稍加改造，在转发请求到后端服务之前与收到响应结果之后，需要记录一下这个阶段的耗时。

并且需要把写入日志的代码逻辑修改为调用监控服务

监控

Zuul已实现连接数相关的统计

关联的Handler类：

- ServerChannelMetrics
  - totalConnections：总连接数
  - connectionClosed：已关闭的连接数
  - connectionErrors：连接失败的数量
  - connectionThrottled：连接拒绝的数量
- PerEventLoopMetricsChannelHandler$Connections
  - 单个EventLoop的连接数
- PerEventLoopMetricsChannelHandler$HttpRequests
  - 单个EventLoop的处理中的请求数

添加定期上报的代码，调用监控监控服务进行数据上报

同时添加代码监控异常exception信息，并且进行数据上报

监控信息的展示

先采用链接跳转的方式，网关管理后台只提供监控页面的链接入口

后续改为调用监控服务的REST API接口读取监控数据，监控的展示内置于网关管理后台

告警

识别到网关时延过大、返回码、请求量突降、异常数量过多，写入日志中心，并调用告警接口进行通知。

告警类型：

- 网关时延过大，超过阈值
- 异常HTTP返回码
- 请求量突降
- 异常数量过多

9.限流/熔断

分布式限流

通过编写Inbound类型的Filter整合vsentinel功能实现分布式限流，限流的配置来源于API级别的治理信息

限流规则

- QPS限流
- 拒绝策略

特别说明：不支持单机限流

熔断

通过编写Inbound & Outbound类型的Filter整合vsentinel功能实现熔断，熔断的配置来源于API级别的治理信息

熔断规则

- 响应时间超过阈值
- 错误率

10.非功能性需求_架构设计

压测(性能、耗时统计)

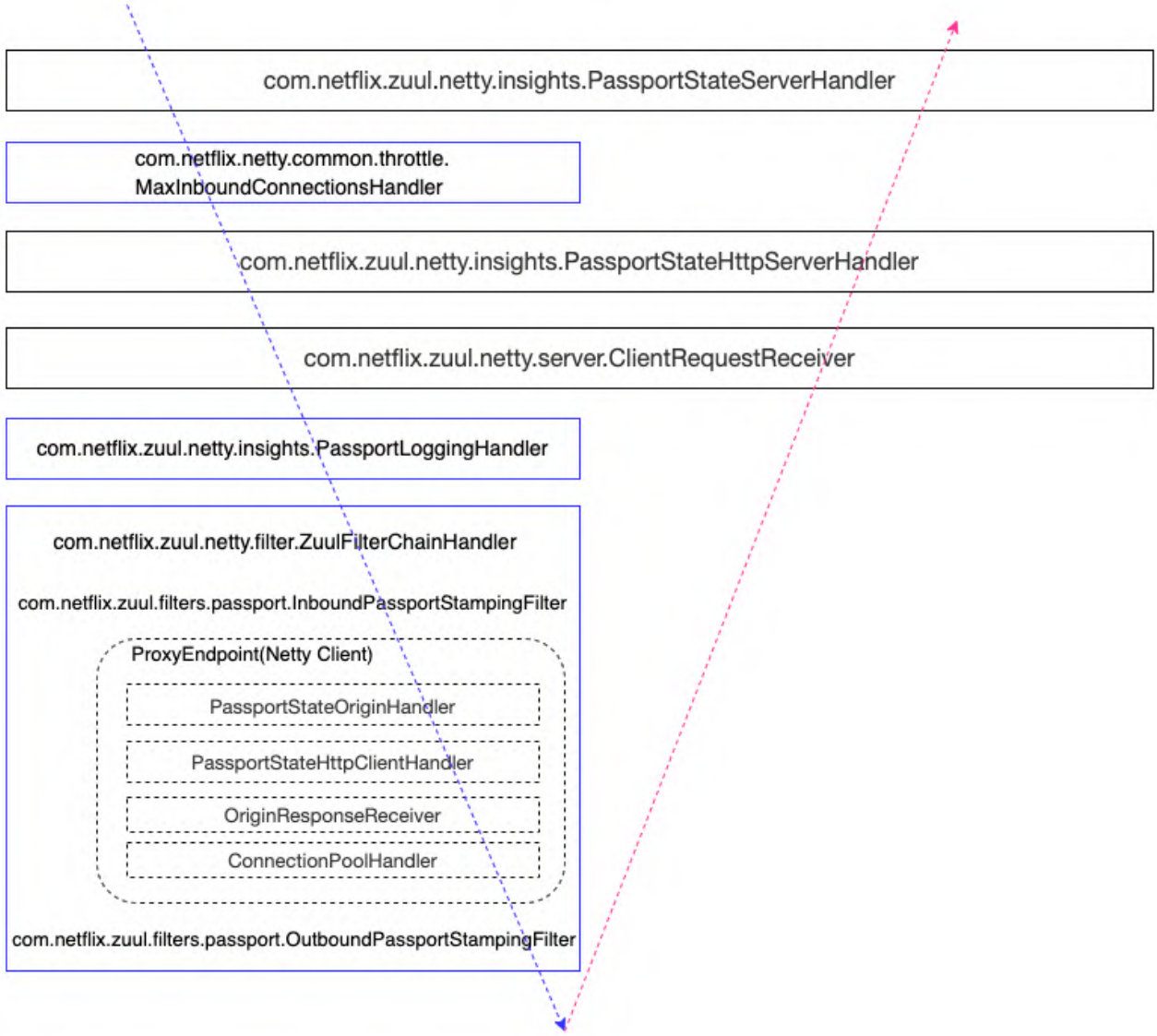
压测工具：wrk压测工具、基于nGrinder的分布式压测平台

性能调优

GC调优

耗时点排查：在Zuul的处理各个阶段进行埋点，输出性能的瓶颈点

Zuul提供了`Request Passport`功能，可以在Netty的Handler、Filter中插入`PassportState`状态点，后续通过监控进行性能排查



系统可用性/健壮性

网关服务、网关管理后台均采用集群化部署

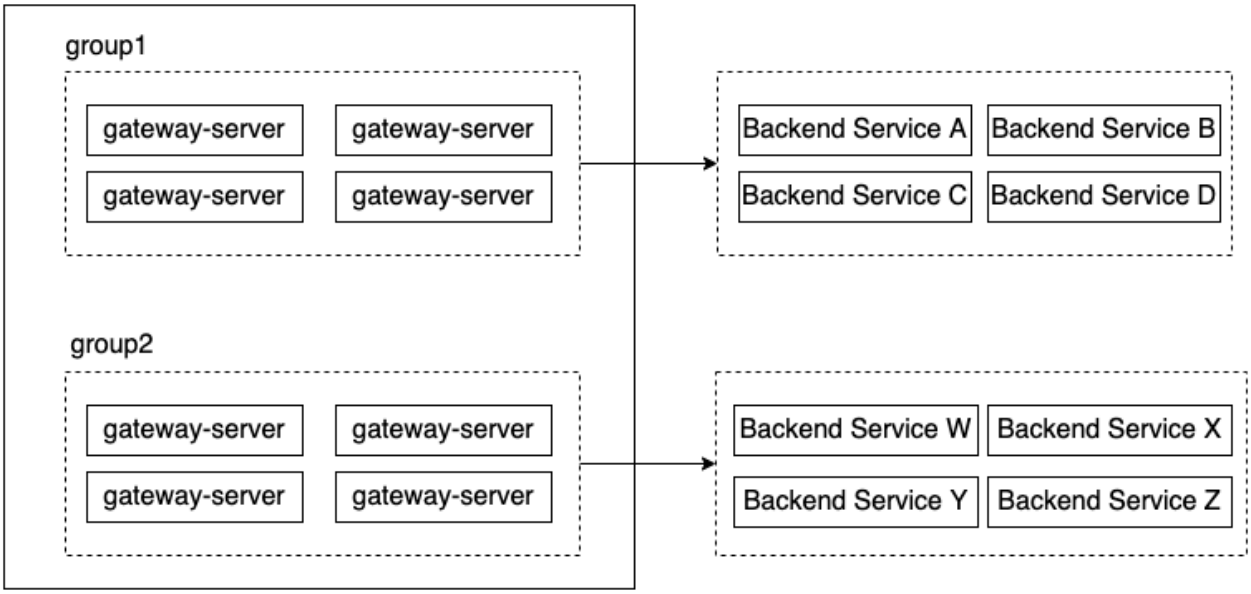
部署新功能、迭代均需在测试环境压测1个礼拜；上线之初，先灰度几台机器运行3天，如果无明显异常，则全量部署

灰度配置下发支持全局配置、应用级别治理配置、API接口级别治理配置、Groovy Filter

网关集群分组隔离

考虑到网关集群可能数量增大，需要对集群进行分组隔离，目前CI/CD不支持分组管理，退而求其次的做法是创建不同的PAAS服务。

因为分组的原因，一个网关服务不一定需要拉取全量的配置与后端应用信息，所以每个网关服务会配置一个groupId，只需要拉取这个分组的配置。



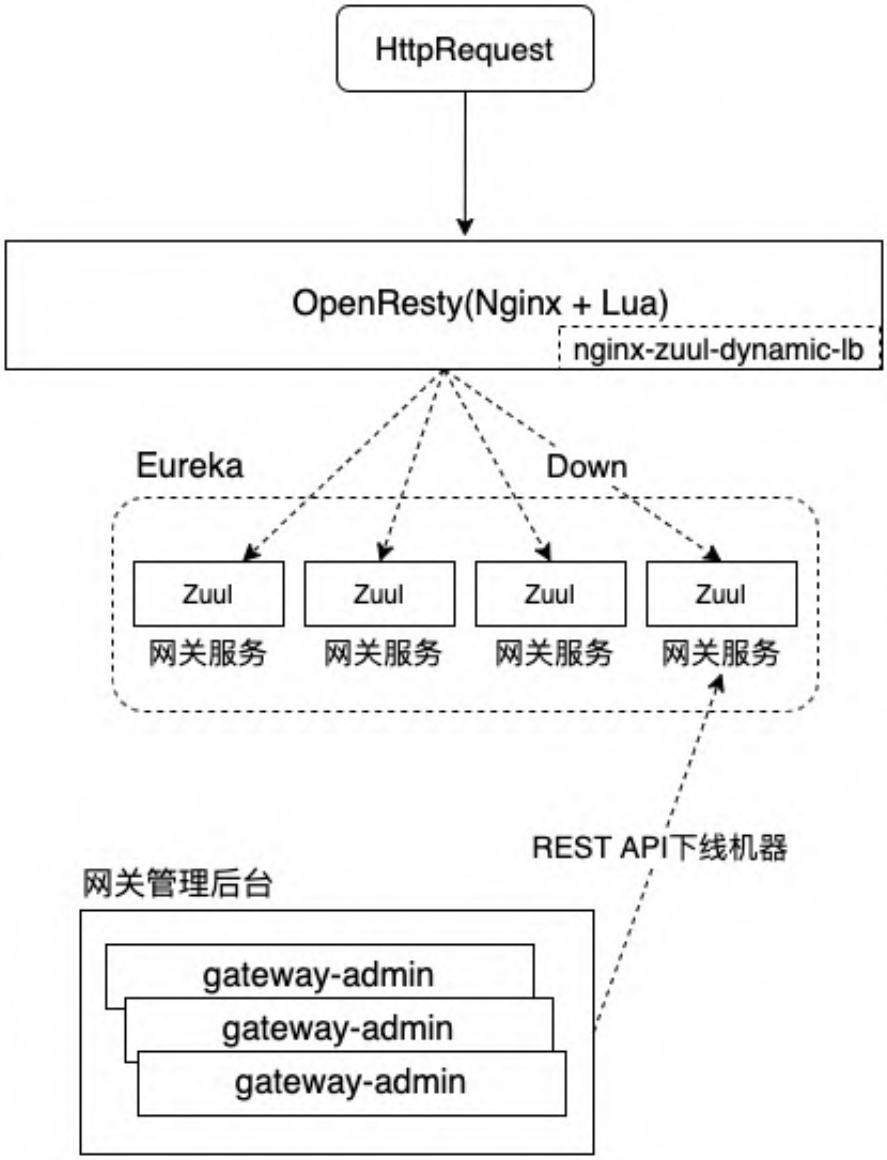
网关集群的扩容/缩容

目前网关暂且不考虑基于k8s、docker部署，走CICD非标准发布流程

但是也需要跟Nginx打通，动态上线、下线网关服务机器，不需要手工修改Nginx转发配置，这样才可以实现部署时，网关流量无损。

业界的开源方案：

https://github.com/SpringCloud/nginx-zuul-dynamic-lb



目前公司级别也已实现**HTTP无损发布**，网关服务发布的时候采用**HTTP无损发布**即可

11.常见问题解答

如何快速迁移?

因为后端服务机器列表支持手动添加，API元数据信息也支持手动添加，可以无缝迁移

网关的分组隔离粒度?

粒度：后端业务应用

网关分组隔离目的在于隔离不同的业务，而不是API；通过分组隔离，保证网关服务的稳定、服务之间的互不影响